IAC-12- B2.2.5

SCAN TESTBED SOFTWARE DEVELOPMENT AND LESSONS LEARNED

Thomas J. Kacpura[1]
NASA Glenn Research Center, U.S.A, Thomas.J.Kacpura@nasa.gov

Denise M. Varga[2]
NASA Glenn Research Center, U.S.A., Denise.M.Varga@nasa.gov

NASA has developed an experimental flight payload, the Space Communication and Navigation (SCaN) Testbed, to investigate software defined radio (SDR) communications, networking, and navigation technologies, and is operational in the space environment. The payload consists of three software defined radios each compliant to NASA's Space Telecommunications Radio System Architecture, a common architecture standard for space software defined radios. These software defined radios are new technology developments for NASA and industry partners. Launched in July 2012, the payload is externally mounted to the International Space Station truss for conducting experiments representative of future mission capability. Experiment operations will include in-flight reconfiguration of the SDR waveform functions and payload networking software. The flight system will communicate with NASA's orbiting satellite relay network, the Tracking and Data Relay Satellite System (TDRSS) at both S-band and Ka-band and direct to the ground to any Earth-based compatible S-band ground station. The system will be available for experiments by industry, academia, and other government agencies to participate in the technology assessments and standards advancements. This paper focuses on software lessons learned through development, integration and testing as related to the avionics processor system, and the software required to command, control, monitor, and interact with the SDRs, as well as the other communication payload elements.

I. INTRODUCTION

THE National Aeronautics and Space Administration (NASA) Space Communications and Navigation (SCaN) Testbed Project is studying the development, testing, and operation of software defined radios (SDRs) and their associated applications for future use by NASA missions. To that end, the NASA Glenn Research Center (GRC) has assembled and launched a flight testbed which consists of reconfigurable and reprogrammable SDRs operating at L-band, S-band, and Ka-band, along with the required radio frequency (RF)/antenna systems necessary for communications. The three SDRs were built by Jet Propulsion Laboratory (JPL)/Cincinnati Electronics, General Dynamics Advanced Information Systems, and the Harris Corporation. The JPL SDR can receive Global Positioning Satellite (GPS) signals while simultaneously operating as an S-band transceiver, and is a heritage design based on the Electra software defined radio. The General Dynamics SDR is capable of full-duplex S-band communications, and leverages GD's experience with the 4th generation Tracking Data and Relay Satellite System (TDRSS) transponder. Harris Corporation provided a full-duplex Ka-band SDR, which is the first NASA Ka-band SDR. All three SDRs are compatible with TDRSS.

The end use of the on-orbit, adaptable, Software Defined Radio (SDR)/Space Telecommunications Radio System (STRS)-based testbed facility is to conduct a suite of experiments on the International Space Station (ISS) to advance technologies, reduce risk, and enable future mission capabilities. The SCAN Testbed will provide NASA, industry, other Government agencies, and academic partners the opportunity to develop and field communications, navigation, and networking technologies in the laboratory and space environment based on reconfigurable SDR platforms and the STRS Architecture. An example of this might be a lunar rover communicating with a home base on the moon, and the home base relaying communications back to earth.

The SDRs are a new technology for NASA, and the support infrastructure they require is different from legacy, fixed function radios. SDRs offer the ability to reconfigure on-orbit communications by changing software for new waveforms and operating systems to enable new capabilities or fix any anomalies, which was not a previous option. Examples are implementing a new coding scheme or a new modulation technique, or simply changing the data rate. These SDRs are not stand alone devices, but required an external source of command and control and data handling. This requires extensive software to be developed to utilize the full potential of these reconfigurable platforms.

This paper focuses on development, integration and testing as related to the avionics processor system, and the software required to command, control, monitor, and interact with the SDRs, as well as the other communication payload elements. An extensive effort was required to develop the flight software and meet the NASA requirements for software quality and safety.

The flight avionics is radiation tolerant, and these processors have limited processing capability in comparison to terrestrial counterparts. A big challenge was interfacing the avionics with multiple SDRs simultaneously, which complicates the effort. The effort also includes ground software, which is a key element for both the command of the payload, and displaying and archiving the telemetry and data created by the payload.

The verification of the software was a time consuming effort. The challenges of specifying a suitable test matrix with reconfigurable systems that offer numerous operating configurations is highlighted. Since the flight system testing requires methodical, controlled testing that limits risk, a nearly identical ground system to the on-orbit flight system was required to develop the software and write verification procedures before it was installed and tested on the flight system. This effort is the basis of a new testing paradigm that goes beyond the current "test as you fly, fly as you test" approach.

The development of the SCAN Testbed was an accelerated effort to meet launch constraints, and this paper discusses tradeoffs made to balance needed software functionality and still maintain the schedule. Future upgrades are discussed that optimize the avionics and allow experimenters to utilize the SCAN Testbed potential.

II. GENERAL LESSONS LEARNED

This section covers some of the general lessons learned in developing the software for this project. Many of the lessons learned were driven by developing software for a flexible testbed with reconfigurable radios for which existing design approaches are no longer suitable.

II.I Balance the "ilities" (flexibility, upgradeability, etc.) Offered by SDRs with Resources and Schedule

The existing approach for procuring radios is to choose ones that meet the existing requirements exactly. SDRs are designed to meet more than the existing launch requirements; they need to be sized with additional resources so that new capabilities can be added. One challenge is that spacecraft often have fixed resources, so additional capabilities need to be carefully sized.

II.II Choose Your Test Matrix Carefully

Mission critical systems such as radio systems require that they are carefully tested to ensure that they will perform as designed under all possible conditions. This is in opposition to the fact that the schedule for developing and testing spacecraft systems is usually very constrained, driven by launch date requirements. When introducing reconfigurable systems, they require testing a larger number of operating conditions than fixed legacy systems. This requires a careful analysis to select the key operating conditions to verify, focusing on understanding the nominal and range of conditions in which the radio is expected to operate. With Software Defined Radios, there can be an infinite number of test cases and properly bounding your test conditions is key to verification prior to operations.

II.III Good SDR Documentation Set Required

Good documentation is key to acquiring an in-depth understanding of the SDRs and their operation. Each of the agreements with the vendors required a set of documentation to be delivered at various portions of their development cycle. The documentation needs to describe the hardware and software in detail, as well as the operation of the radio. Any commercial software systems need to have documentation provided. The interfaces that the spacecraft will connect to the radio needs to be well-documented, and having this information early in the development cycle allows parallel development of the avionics with its data systems and the radios. Vendor test data is also useful to understanding the performance and operation of the SDR, along with operating logs and anomaly resolution data.

II.IV High Fidelity Software Development Systems Are Necessary

The project requested breadboard, engineering model and flight models of each radio. Since each radio had a fair amount of development, the goal was to have a breadboard first, the engineering model next, and the flight system as the final deliverable. The breadboard systems were early development systems to allow interfacing with the avionics, with a focus on the processor and waveform signal processing memory. The engineering model was to be nearly identical to the fight system, to be used in the ground integration unit for flight system software development and hardware checkout. The use of these interim SDR deliveries would enable the integration and test of the flight unit radios to be proven before installation into the flight payload.

The challenge was that the actual deliveries did not occur quite that way. The breadboard systems were early development systems, but in the effort to allow an early vendor delivery, they lacked the fidelity of the engineering and flight units, most importantly the RF functionality. Having the ability to upgrade these units would have been useful post-launch for future waveform development. Also, the engineering units and flight units were delivered very close together, leaving little to no time to incorporate findings from testing with the engineering models into the flight radios. New developments often run into issues;

schedule risks that were realized resulted in close deliveries of the SDRs leaving little or no opportunity to incorporate changes from engineering model testing into the flight models.

II.II Engineering and flight units must be identical.

The engineering units used similar components to the flight units for the most part, but they were not identical. The trade was cost and schedule, since the flight units used radiation hardened or tolerant components with long lead times. However, using identical components is crucial in the engineering systems, so that software with critical timing margins and performance can be verified on the ground before being deployed on the flight unit. For example, the firmware executed on Xilinx Virtex IV FPGAs has different performance due to timing differences between the space grade and commercial grade components. These timing differences are critical when developing high data rate waveforms that have timing margin constraints, and the ground testing is not a good predictor of the flight performance. Another example is the first time we took our avionics software from the Ground Integration Unit (GIU), which is our ground testbed, to the flight system, the boot process started, then almost immediately the avionics computer rebooted. This cycle repeated over and over until power was removed from the system. It turns out that the timing of the commercial hardware was different enough from the flight hardware that on the flight system, when the SpaceWire card was initialized, it immediately began sending out interrupts before the interrupt handler had been initialized. This had never been a problem on the GIU. The solution was to initialize the interrupt handler before initializing the SpaceWire card.

II.V Prioritize Your Requirements – They Are Not All Created Equal

Due to a compressed development schedule, the software team prioritized requirements to ensure that the capabilities absolutely required to operate the testbed were implemented first. This way, if unforeseen problems impacted the schedule, the payload could still be shipped in time for launch. Since this is a "reconfigurable" testbed, the ability to upload new software is mandatory. The payload also has safety-critical software to inhibit radiation when it presents a hazard (during Extra-Vehicular Activity or docking operations by visiting vehicles, as two examples), and these inhibits must be verified in order to fly on ISS. Finally, the interface to ISS is necessary in order to be able to command the payload and to upload and download files, including new software. These requirements became Priority 1 (P1) requirements. Other requirements necessary to meet the project's "Minimum Success" criteria, or which required the actual flight hardware to verify were also categorized as P1. Priority 2 (P2) requirements were those requirements that were strongly tied to meeting the remaining Mission Success criteria. P3 requirements were those tied to experiment capabilities that could be uploaded after launch without impacting payload checkout.

This categorization proved invaluable when negotiating requirements implementation within the project. At launch time, all P1, several P2, and one P3 requirements were implemented. Several large capabilities were not implemented, but are scheduled for development post-ship and are currently under development.

II.VI GIU: Proved to be invaluable as a tool to dry run procedures, unit test software, and perform software verifications: Saved valuable time on the critical path of Flight System

As described above, the engineering unit radios were installed into the GIU. The development and testing conducted on the GIU was a heavily-scheduled, but important, development tool before conducting the integration and testing of the flight system. Operation on the flight system required prior verification on the GIU with the engineering units and an approved procedure, so considerable time was spent using the GIU to prepare for flight system operation. The contention for the use of this hardware required the different project organizations, e.g. Comm, Software, Operations, to schedule shifts of time on the GIU from 6 am to midnight on some days. The breadboards were also used for development, but they lacked the fidelity of the GIU to go directly to flight unit operation.

Software verification was also conducted on the GIU. While the key software tests were conducted on the flight system, an exercise was done to identify software verifications that could be conducted on the GIU instead of the flight system with relatively low risk of seeing differences between the two systems. This allowed a parallel path to complete software verifications, but testing had to be carefully considered so verifications done on the GIU were valid as flight demonstrations.

III. SOFTWARE LESSONS LEARNED

The following lessons apply specifically to software development.

III.I DRIVE YOUR OWN INTERFACES: If you don't drive your interfaces, they will be driven for you

The software defined radio vendors reused and modified as much software as possible from previous efforts. Therefore, many of the interfaces and command dictionaries were already in place. This required the

avionics development to generate three different command processors. Similarly, each radio had its own set of telemetry with little commonality among them. This reduced development time in the radios, but increased time and complexity in the avionics development.

Each radio has two separate interfaces – one used for commanding and telemetry and one used for data transfer.

Command and Telemetry Interface

Each SDR has a separate command format. The Harris SDR uses RMAP (remote memory access protocol) command format over SpaceWire with data in 251-byte packets over SpaceWire. The JPL SDR uses a character command format over 1553 with data in 260-byte packets over SpaceWire. The GD SDR uses a fixed binary command format over 1553 with data in 256-byte packets over SpaceWire.

The GD SDR utilizes a fixed binary telemetry packet that is pushed out over 1553 bus at 1 Hz. Avionics must read this data before it is refreshed or it is lost. Harris uses a telemetry packet that is filled by reading the selected parameters from a configuration file and is returned in response to a query by avionics. The SDR JPL has a "heartbeat" telemetry packet that is filled with values of parameters identified in a configuration file, and written to the 1553 registers at 1 Hz. Additional telemetry for the JPL radio can be queried using a string command with parameters indicating the values to be returned. This data is displayed in a serial interface display.

All formats were selected based upon heritage developments. The JPL team focused on flexibility at the expense of a high level of overhead. The GD team created a very fixed format which was efficient but inflexible to changes. Harris selected an approach that was in-between with a fixed amount of space in packet, but with the ability to modify contents.

Data Interface

SpaceWire was selected as the data interface based on recommendations from radio partners for a solid, high-performance space data application. This hardware and software technology was new to the avionics development team.

The data interface had to be independently developed for each radio. The data is sent in a framed format, with a synchronization marker and transfer frame primary header (TFPH) to lead a fixed block of data. In addition, the SpaceWire protocol has a SpaceWire header. The Harris SDR adds and removes the synchronous marker and the SpaceWire header, leaving the avionics to simply grab the data sent through without any headers. The GD SDR adds and removes the TFPH but not synchronization marker. Avionics is required to add and remove the synchronization marker for data transfer. The JPL SDR data interface did not change the sync marker or the TFPH. Avionics was required to add and remove both of these for JPL data transfer.

Ground Displays

Each radio had distinct requirements for display of its telemetry resulting in different layouts and number of screens on the ground workstations. The GD telemetry stream contained both OE (operating environment) and waveform data together. Harris required multiple screens for just its waveform data. Ground displays could not be reused between radios and thus we had 30 or more displays to generate and maintain.

Some data is displayed periodically (1 Hz telemetry) whenever the payload or radio is on. There is also the need to display graphs and plots of real time data over time. One radio requires a screen to display character commands and command responses being transferred over the serial interface to the radio.

A theme that was common throughout our development was – Do you pay up front to have commonality, or do you cut costs to vendors and incur more cost for payload development? In our case, since the payload was a new development and the radios were mostly redesigns of existing models, the payload development took the hit to meet the radio interfaces.

III.II COMM SYSTEMS ENGINEER: To help flow requirements down to subsystems

The ability to take system level requirements and flow them down to each subsystem is a key aspect in designing and verifying the system. It is also imperative to define the interfaces between each subsystem, as well the external interfaces.

One of the key tools used to extract the lower level requirements and define the interfaces is a data flow diagram. A data flow diagram is a graphical representation of the flow of data, and shows what kinds of information will be input to and output from the system, where the data will come from and go to, and where the data will be stored. For the SCaN Testbed, the diagram shows the data flow originating on the ground, flowing up to the flight system, then back down to the ground again.

In particular, the data flow passing through the payload needed to be developed in detail. The data size and headers change as the data passes through the various segments of the flight and ground system, which is different for each radio and impacts the processing required. Example questions the team asked to develop the diagram included: "What does 100 Mbps data rate mean from radio to ground?" "What is the data flow from each radio to the avionics?" Another factor examined was the impact of using forward error correction coding schemes, which effectively doubled the data rate for the standard ½ rate coding scheme used. These diagrams were essential to understand the format and the data rate, and many diagrams were developed to share this insight between the communications and the software team, each of which had a different perspective on the data definitions.

Another complex issue for resolution was allocating time synchronization from higher level requirements down to each of the lower level subsystems. Time synchronization is a key feature required so that the SCaN Testbed is on the same time reference as the ISS and the TDRSS. The flight system has the potential of several different sources (ISS, TDRSS, GPS) to obtain the timing, and each source requires a different method of transferring this data to the payload. Also, the avionics and each SDR need to be synchronized. This is to ensure that all the commands, telemetry, and data are on the same time reference so these parameters can be correlated and stored data can be understood. This sounds like a simple task, but was not simple to implement. The reason for this is that the avionics and each radio had a different time oscillator with a different accuracy. Also, the choice of the master clock used to keep the master synchronization could change depending on the accuracy required. Measuring the accuracy itself was problematic, since your measurement system needs to have a higher precision than what you are measuring, and we were measuring the most precise portion of our system. We didn't have a convenient way of measuring this and are still working on this post-ship.

One lesson learned was with the external ground support test equipment used to verify the operation of the system. A large amount of data was generated that needed to be evaluated post-test, and the project needed to be convinced that the effort to synchronize all the external equipment was necessary and the impact to the critical path schedule was essential. Another lesson learned was that the time keeping software of all systems needed to be understood and carefully monitored. During the thermal vacuum test, one of the critical ground support computers switched to European daylight savings time. The reason for this change was unknown, but the impact of this unexpected change was large in correlating all the collected data.

III.III Allow Additional Tine and Money When Integrating Hardware From Different Vendors

Our avionics computer is custom-built with cards from various vendors. Some cards were sold to us by Vendor A, but portions of the card or drivers were developed by Vendor B. When problems arose with the hardware and software integration, it was extremely difficult to pinpoint what component was causing the problem. Each of the vendors believed that their product was accurate and the adjoining part was at fault. This caused extreme schedule delays through attempts to communicate with all possible vendors, finding ways to recreate the problem for them in order to get their help, then working with them for a solution once we got their attention. The vendors were willing to assist us and were invested in our success. However, our software engineers bore the burden of much unplanned time tracking down hardware issues and isolating the cause, with the vendors' support. In some instances, our development time was TRIPLE what we had planned. This time should be anticipated and planned for when integrating systems with hardware from different vendors.

One of our internal interfaces proved particularly problematic. This was the interface between the SpaceWire card and the main processor in the computer, which relied on communications over the PCI bus. Each vendor had interpreted the PCI bus specification in a slightly different way with respect to polling of direct memory access (DMA). This difference of interpretation presented itself in our system as a deadlock condition over bus usage when attempting to transfer high data rates between avionics and the radios. The payload would grind to a halt and our only option was to reboot the entire system. In this case it was difficult to identify which vendor was "right" since we were dealing with a specification that was, perhaps, not as specific as it needed to be. The deciding factor was which component cost us less schedule time to fix. One component required that we send a computer card back to the vendor to have them replace a field-programmable gate array (FPGA). The other component could be reprogrammed at our site, which was far less impact to our schedule.

In the end, we did have to replace the firmware on a card that provided our communications over MIL-STD-1553 and our digital input/output functions. We originally purchased components to develop two flight computers and two development computers. Nine months later we decided to add two more development systems to support our development schedule and twelve software developers and we purchased additional components. In putting together our additional development systems, we could not get the 1553 boards to boot in the new systems. It turns out that these newer

boards were a complete redesign of the original boards, although they had the same model number. We could not use these new boards without compiling new drivers and maintaining two separate software loads, so we chose to make do with just our older boards. With about six months to go before shipping our payload, we discovered that the older boards were the culprit for spurious digital input/output signal changes that were NOT initiated through software. This affected our safety-critical ability to make sure or payload was not radiating (transmitting) during certain times where the radiation could harm people or equipment. It seems we had discovered the reason for the sudden redesign of these boards. We sent all of our older boards back to the vendor to replace the firmware and upgrade them to the newer design. New software drivers had to be written and integrated into the code base. During the transition, software testing required switching between the old and new code base, depending on what type of hardware was in the system under test. In order to keep testing going, the cards were sent back incrementally so only one system was down at a time.

III.IV NEW TECHNOLOGY: When adopting new technologies (SpaceWire) have a direct line to the experts around the world

We had a technical scenario that had never been done before as far as we knew - three SpaceWire interfaces controlled by a single computer with four different SpaceWire implementations. With the problems described in paragraph III.III, our software development took about four times what we had estimated.

Repairs were done in parallel with other testing that didn't require SpaceWire data transfer to minimize loss of schedule. The radios had a self-test functionality that would generate data to transmit and calculate bit error rate on data being received, which was used instead of flowing data through avionics while we struggled to solve the SpaceWire issues. We pulled in experts from Goddard Space Flight Center and followed their best practices for other spacecraft.

GRC put together a Tiger Team of NASA and vendor consultants to solve the SpaceWire problems. The team first looked at software-only solutions; but after further investigation and determination of critical root cause(s), worked with the vendor to change firmware. The software team rewrote the drivers to adapt to the changes. The entire avionics unit was pulled to access firmware and new environmental testing had to be completed. Analysis showed FPGA change did not impact system Electro-Magnetic Interference results. Throughout the year of SpaceWire issues, functional and environmental testing was conducted with incremental SpaceWire performance capabilities

Take-away:

SpaceWire performance is great, but buyers beware. SpaceWire hardware is not robust and firmware/software interface standards are not mature. **Utilize FPGAs that are reliably reprogrammable without removal from system.**

V. CONCLUSION

The development of any complex system requires a well-constructed plan to be successful. Spaceflight systems have been developed to extensive standards and practices to insure mission critical systems such as radios and control avionics operate as expected once launched. The introduction of reconfigurable SDRs requires an adjustment of these practices to account for the flexibility that the systems offer, but still provide confidence in proper operation.

The lessons learned from this development should be applicable to future spaceflight systems with reconfigurable components.

.

VI. ACRONYMS

Direct Memory Access (DMA)
Field-Programmable Gate Array (FPGA)
General Dynamics Corporation (GD)
Glenn Research Center (GRC)
Global Positioning System (GPS)

Goddard Space Flight Center (GFSC)
Ground Integration Unit (GIU)
International Space Station (ISS)
Jet Propulsion Laboratory (JPL)
National Aeronautics and Space Administration (NASA)
Operating Environment (OE)
Radio Frequency (RF)
Software Defined Radio (SDR)
Space Communications and Navigation (SCaN)
Tracking and Data Relay Satellite System (TDRSS)
Transfer Frame Primary Header (TFPH)

VII. REFERENCES

Reinhart, R., Kacpura, T., Handler, L., Hall, C., Mortensen, D., Johnson, S., et. al. "Space Telecommunications Radio System (STRS) Architecture Standard, Release 1.02.1. NASA/TM-2010-216809, NASA Glenn Research Center, Cleveland OH, December 2010.

[1] SCaN Testbed Communications System Lead, AIAA Non-member
[2]SCaN Testbed Lead Software Engineer, AIAA Non-member